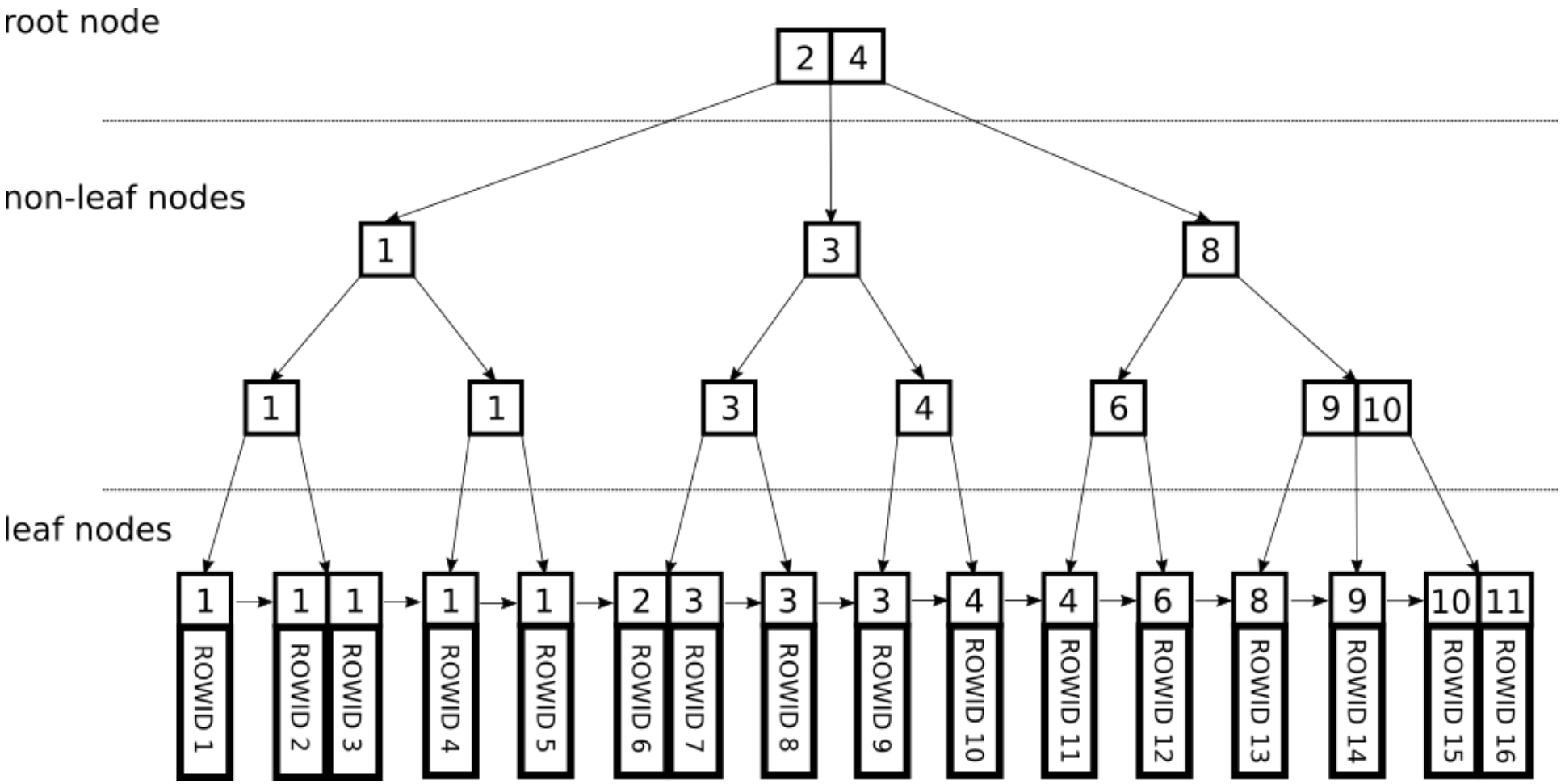
An introduction to Postgres Join, Sort & GroupBy

Sebastian Brestin – Data Engineer @ qwertee.io
28 February 2020

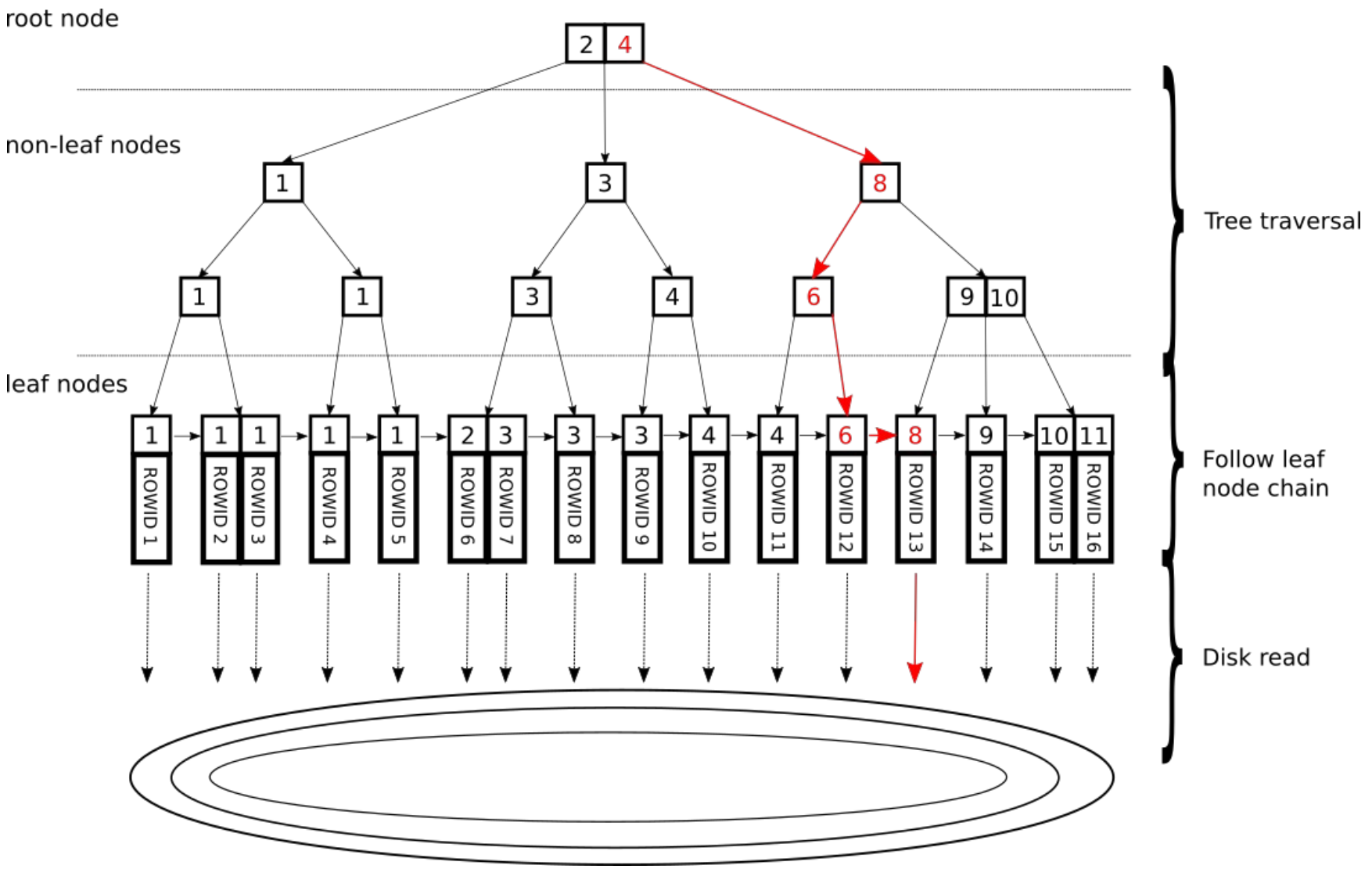
Content

- B-tree Index
- Join
- Sort
- Group By

B-tree Index Structure



Index Lookup



Tables

```sql
CREATE TABLE "public"."addresses" (
    "id" integer NOT NULL,
    "city" character varying(40) NOT NULL,
    "country" character varying(40) NOT NULL,
    "street" character varying(40) NOT NULL
);

CREATE TABLE "public"."employees" (
    "id" integer NOT NULL,
    "company_id" integer NOT NULL,
    "dep" integer NOT NULL,
    "first_name" character varying(20),
    "last_name" character varying(20),
    "salary" integer,
    "address_id" integer
);
```

100k records

Join Types

- Nested Loops Join
  - The right relation is scanned once for every row found in the left relation
- Hash Join
  - The right relation is first scanned and loaded into a hash table
- Merge Join
  - Each relation is sorted on the join attributes before the join starts

Nested Loops Join

```
EXPLAIN ANALYZE SELECT * FROM employees AS e JOIN addresses AS a ON e.address_id = a.id LIMIT 1;
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.00..2184.04 rows=1 width=63) (actual time=31.105..31.108 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..218403817.00 rows=100000 width=63) (actual time=31.103..31.103 rows=1 loops=1)
         Join Filter: (e.address_id = a.id)
         Rows Removed by Join Filter: 33951
         ->  Seq Scan on employees e  (cost=0.00..1836.00 rows=100000 width=36) (actual time=0.018..0.018 rows=1 loops=1)
         ->  Materialize  (cost=0.00..2915.00 rows=100000 width=27) (actual time=0.013..21.887 rows=33952 loops=1)
               ->  Seq Scan on addresses a  (cost=0.00..1731.00 rows=100000 width=27) (actual time=0.010..7.458 rows=33952 loops=1)
 Planning time: 0.310 ms
 Execution time: 32.010 ms
(9 rows)
```

Without index, top 1

Nested Loops Join

```
EXPLAIN ANALYZE SELECT * FROM employees AS e JOIN addresses AS a ON e.address_id = a.id LIMIT 100;
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=3665.00..3670.46 rows=100 width=63) (actual time=73.465..73.549 rows=100 loops=1)
   ->  Hash Join  (cost=3665.00..9124.00 rows=100000 width=63) (actual time=73.464..73.539 rows=100 loops=1)
         Hash Cond: (e.address_id = a.id)
         ->  Seq Scan on employees e  (cost=0.00..1836.00 rows=100000 width=36) (actual time=0.075..0.092 rows=193 loops=1)
         ->  Hash  (cost=1731.00..1731.00 rows=100000 width=27) (actual time=72.471..72.471 rows=100000 loops=1)
               Buckets: 65536  Batches: 2  Memory Usage: 3448kB
               ->  Seq Scan on addresses a  (cost=0.00..1731.00 rows=100000 width=27) (actual time=0.021..31.573 rows=100000 loops=1)
 Planning time: 0.458 ms
 Execution time: 74.150 ms
(9 rows)
```

Without index, increase limit

Nested Loops Join

CREATE INDEX ON employees (address_id);

```
EXPLAIN ANALYZE SELECT * FROM employees AS e JOIN addresses AS a ON e.address_id = a.id LIMIT 100;
                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.29..41.22 rows=100 width=63) (actual time=0.034..0.868 rows=100 loops=1)
   ->  Nested Loop  (cost=0.29..40929.00 rows=100000 width=63) (actual time=0.032..0.820 rows=100 loops=1)
         ->  Seq Scan on addresses a  (cost=0.00..1731.00 rows=100000 width=27) (actual time=0.016..0.050 rows=102 loops=1)
         ->  Index Scan using employees_address_id_idx on employees e  (cost=0.29..0.37 rows=2 width=36) (actual time=0.004..0.006 rows=1 loops=102)
               Index Cond: (address_id = a.id)
 Planning time: 0.431 ms
 Execution time: 0.945 ms
(7 rows)
```

Hash Join

CREATE INDEX ON employees (address_id);

```
EXPLAIN ANALYZE SELECT * FROM employees AS e JOIN addresses AS a ON e.address_id = a.id;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=3665.00..9124.00 rows=100000 width=63) (actual time=52.290..110.064 rows=100000 loops=1)
   Hash Cond: (e.address_id = a.id)
   ->  Seq Scan on employees e  (cost=0.00..1836.00 rows=100000 width=36) (actual time=0.010..5.149 rows=100000 loops=1)
   ->  Hash  (cost=1731.00..1731.00 rows=100000 width=27) (actual time=51.542..51.542 rows=100000 loops=1)
         Buckets: 65536  Batches: 2  Memory Usage: 3449kB
         ->  Seq Scan on addresses a  (cost=0.00..1731.00 rows=100000 width=27) (actual time=0.007..12.274 rows=100000 loops=1)
 Planning time: 0.388 ms
 Execution time: 112.695 ms
(8 rows)
```

Merge Join

CREATE INDEX ON employees (address_id);
ALTER TABLE addresses ADD PRIMARY KEY (id);

```
EXPLAIN ANALYZE SELECT * FROM employees AS e JOIN addresses AS a ON e.address_id = a.id LIMIT 10000;
                                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.62..1079.20 rows=10000 width=63) (actual time=0.036..30.765 rows=10000 loops=1)
   ->  Merge Join  (cost=0.62..10786.40 rows=100000 width=63) (actual time=0.034..29.044 rows=10000 loops=1)
         Merge Cond: (e.address_id = a.id)
         ->  Index Scan using employees_address_id_idx on employees e  (cost=0.29..5948.29 rows=100000 width=36) (actual time=0.016..11.435 rows=10000 loops=1)
         ->  Index Scan using addresses_id_idx on addresses a  (cost=0.29..3338.29 rows=100000 width=27) (actual time=0.012..5.387 rows=13858 loops=1)
 Planning time: 0.443 ms
 Execution time: 31.726 ms
(7 rows)
```

With 2 indexes

Sort

CREATE INDEX ON employees (address_id);

```
1  EXPLAIN (ANALYZE) SELECT first_name FROM employees ORDER BY address_id;
2                                                     QUERY PLAN
3  ------------------------------------------------------------------------------------------------------------------------------------------------
4   Index Scan using employees_address_id_idx on employees  (cost=0.29..5948.29 rows=100000 width=12) (actual time=0.027..64.605 rows=100000 loops=1)
5   Planning time: 0.106 ms
6   Execution time: 70.229 ms
7  (3 rows)
8
```

ASC, with simple index

Sort

CREATE INDEX ON employees (address_id);

```
1  EXPLAIN (ANALYZE) SELECT first_name FROM employees ORDER BY address_id DESC;
2                                                          QUERY PLAN
3  ------------------------------------------------------------------------------------------------------------------------------------------------------
4   Index Scan Backward using employees_address_id_idx on employees  (cost=0.29..5948.29 rows=100000 width=12) (actual time=0.030..61.170 rows=100000 loops=1)
5   Planning time: 0.169 ms
6   Execution time: 66.098 ms
7  (3 rows)
8
```

DESC, with simple index

Sort

DROP INDEX employees_address_id_idx;

```
EXPLAIN (ANALYZE) SELECT first_name FROM employees ORDER BY address_id DESC;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=10140.82..10390.82 rows=100000 width=12) (actual time=87.877..100.261 rows=100000 loops=1)
   Sort Key: address_id DESC
   Sort Method: external merge  Disk: 2336kB
   ->  Seq Scan on employees  (cost=0.00..1836.00 rows=100000 width=12) (actual time=0.018..23.174 rows=100000 loops=1)
 Planning time: 0.172 ms
 Execution time: 105.352 ms
(6 rows)
```

Without index

Sort

CREATE INDEX ON employees (company_id, dep, last_name);

```
EXPLAIN (ANALYZE) SELECT first_name FROM employees WHERE company_id > 10 ORDER BY company_id ASC, dep ASC;
                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using employees_company_id_dep_id_idx on employees  (cost=0.42..6311.51 rows=89780 width=16) (actual time=0.030..68.143 rows=89900 loops=1)
   Index Cond: (company_id > 10)
 Planning time: 0.161 ms
 Execution time: 74.346 ms
(4 rows)

```

With multi index

Sort

CREATE INDEX ON employees (company_id, dep, last_name);

```
EXPLAIN (ANALYZE) SELECT first_name FROM employees WHERE company_id > 10 ORDER BY company_id DESC, dep ASC;
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=9472.25..9696.70 rows=89780 width=16) (actual time=114.523..127.646 rows=89900 loops=1)
   Sort Key: company_id DESC, dep
   Sort Method: external merge  Disk: 2456kB
   ->  Seq Scan on employees  (cost=0.00..2086.00 rows=89780 width=16) (actual time=0.016..27.883 rows=89900 loops=1)
         Filter: (company_id > 10)
         Rows Removed by Filter: 10100
 Planning time: 0.147 ms
 Execution time: 132.544 ms
(8 rows)
```

With multi index, bad ordering

GroupBy Types

- Hash algorithm
- Sort/Group algorithm

GroupBy

DROP INDEX ON employees (address_id);

```
EXPLAIN (ANALYZE) SELECT COUNT(*) FROM employees GROUP BY address_id LIMIT 100;
                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=2336.00..2337.00 rows=100 width=12) (actual time=57.438..57.459 rows=100 loops=1)
   ->  HashAggregate  (cost=2336.00..2870.40 rows=53440 width=12) (actual time=57.436..57.451 rows=100 loops=1)
         Group Key: address_id
         ->  Seq Scan on employees  (cost=0.00..1836.00 rows=100000 width=4) (actual time=0.017..9.717 rows=100000 loops=1)
 Planning time: 0.140 ms
 Execution time: 60.030 ms
(6 rows)
```

Without index, using Hash algorithm

GroupBy

CREATE INDEX ON employees (address_id);

```
EXPLAIN (ANALYZE) SELECT COUNT(*) FROM employees GROUP BY address_id LIMIT 100;
                                                    QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.29..7.04 rows=100 width=12) (actual time=0.134..0.323 rows=100 loops=1)
   ->  GroupAggregate  (cost=0.29..3644.53 rows=54024 width=12) (actual time=0.132..0.298 rows=100 loops=1)
         Group Key: address_id
         ->  Index Only Scan using employees_address_id_idx on employees  (cost=0.29..2604.29 rows=100000 width=4) (actual time=0.120..0.164 rows=151 loops=1)
               Heap Fetches: 0
 Planning time: 0.392 ms
 Execution time: 0.398 ms
(7 rows)
```

With simple index, using Sort/Group algorithm

GroupBy

CREATE INDEX ON employees (company_id, dep, last_name);

```
EXPLAIN (ANALYZE) SELECT COUNT(*) FROM employees GROUP BY company_id, dep LIMIT 5;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.42..9.97 rows=5 width=16) (actual time=0.133..0.229 rows=5 loops=1)
   ->  GroupAggregate  (cost=0.42..3822.42 rows=2000 width=16) (actual time=0.131..0.225 rows=5 loops=1)
         Group Key: company_id, dep
         ->  Index Only Scan using employees_company_id_dep_id_idx on employees  (cost=0.42..3052.42 rows=100000 width=8) (actual time=0.074..0.137 rows=240 loops=1)
               Heap Fetches: 0
 Planning time: 14.909 ms
 Execution time: 0.285 ms
(7 rows)
```

```
EXPLAIN (ANALYZE) SELECT COUNT(*) FROM employees GROUP BY dep LIMIT 5;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=2336.00..2336.05 rows=5 width=12) (actual time=50.567..50.570 rows=5 loops=1)
   ->  HashAggregate  (cost=2336.00..2336.20 rows=20 width=12) (actual time=50.566..50.567 rows=5 loops=1)
         Group Key: dep
         ->  Seq Scan on employees  (cost=0.00..1836.00 rows=100000 width=4) (actual time=0.014..12.456 rows=100000 loops=1)
 Planning time: 0.170 ms
 Execution time: 50.625 ms
(6 rows)
```

Index prefix matters

GroupBy

```
1  CREATE INDEX ON employees (company_id, dep DESC, last_name);
2
3  EXPLAIN (ANALYZE) SELECT COUNT(*) FROM employees GROUP BY company_id, dep LIMIT 5;
4                                                        QUERY PLAN
5  ----------------------------------------------------------------------------------------------------------------------------
6   Limit  (cost=2586.00..2586.05 rows=5 width=16) (actual time=45.558..45.560 rows=5 loops=1)
7     ->  HashAggregate  (cost=2586.00..2606.00 rows=2000 width=16) (actual time=45.556..45.557 rows=5 loops=1)
8           Group Key: company_id, dep
9           ->  Seq Scan on employees  (cost=0.00..1836.00 rows=100000 width=8) (actual time=0.013..9.774 rows=100000 loops=1)
10  Planning time: 0.519 ms
11  Execution time: 45.661 ms
12 (6 rows)
13
```

Bad index order

More

- https://www.postgresql.org/docs/10/planner-optimizer.html
- SQL Performance explained - Markus Winand
- PostgreSQL 10 High Performance, Chapter 10 - Query Optimization - Ibrar Ahmed, Gregory Smith, Enrico Pirozzi
- https://www.postgresql.org/docs/10/indexes-ordering.html
- https://www.qwertee.io/blog/postgresql-b-tree-index-explained-part-1/

Thank you